CF2-extensions as Answer-set Models

Mauricio Osorio [a,1]
[a] *CENTIA*

Juan Carlos Nieves and Ignasi Gómez-Sebastià [b,2]
[b] *Knowledge Engineering and Machine Learning Group*

Abstract. Extension-based argumentation semantics have shown to be a suitable approach for performing practical reasoning. Since extension-based argumentation semantics were formalized in terms of relationships between atomic arguments, it has been shown that extension-based argumentation semantics based on admissible sets such as stable semantics can be characterized in terms of answer sets. In this paper, we present an approach for characterizing SCC-recursive semantics in terms of answer set models. In particular, we will show a characterization of CF2 in terms of answer set models. This result suggests that not only extension-based argumentation semantics based on admissible sets can be characterized in terms of answer sets; but also extension-based argumentation semantics based on Strongly Connected Components can be characterized in terms of answer sets.

Keywords. Argumentation Theory, Answer-set Semantics, Extension-based Argumentation Semantics.

1. Introduction

Although several approaches have been proposed for capturing representative patterns of inference in argumentation theory, Dung's approach, presented in [8], is a unifying framework which has played an influential role on argumentation research and Artificial Intelligence (AI). The kernel of Dung's framework is supported by the following extension-based argumentation semantics (also called *abstract argumentation semantics*): *grounded*, *stable*, *preferred* and *complete* semantics. Even though each of these argumentation semantics represents different patterns of selection of arguments all of them are based on the concept of an *admissible set*. An admissible set can be regarded as a coherent point of view from a conflicting set of arguments. When Dung introduced his argumentation approach, he proved that some extension-based argumentation semantics can be regarded as a special form of logic programming with *negation as failure*. This result defines a general method for generating metainterpreters for argumentation systems as well as a general method for studying the *abstract argumentation semantics*' properties in terms of logic programming semantics' properties. It is worth mentioning that following Dung's approach, other authors have defined additional argumentation seman-

[1]Universidad de las Américas, Sta. Catarina Mártir, Cholula, Puebla, 72820 México. E-Mail:**{osoriomauri@googlemail.com**

[2]Departament de Llenguatges i Sistemes Informàtics. Universitat Politècnica de Catalunya (UPC) C/Jordi Girona 1-3, E-08034 , Barcelona, Spain. E-Mail:**{jcnieves,igomez}@lsi.upc.edu**

tics, such as the semi-stable semantics [4], ideal-semantics [9] and the SCC-recursive semantics [2].

Extension-based argumentation semantics based on admissible sets have shown unexpected behaviors when computing some argumentation frameworks (for instance, frameworks presenting odd-length cycles) [2, 17]. In these cases approaches such as the SCC-recursive semantics [2] come in handy. This approach is based on splitting an argumentation framework in strongly connected components (obtaining sets of arguments whose status does not depend on the status of arguments outside the set) and processing them separately. An order (known as *directionality principle*) is defined between the resulting strongly connected components. Given two strongly connected components SCC_A and SCC_B it is said that SCC_A precedes SCC_B if there is a directed path in $\langle AR, attacks \rangle$ from any argument in SCC_A to an argument in SCC_B. Therefore, the value of (at least) one argument in SCC_B depends on the status of (at least) one argument in SCC_A. The processing of the strong connected components is performed in a recursive way, where the results of one strong connected component affect the following ones. This approach is remarkable for two reasons: a) it is powerful, as it is able to properly process semantics on argumentation frameworks where other approaches do not show proper behaviors b) it is versatile, on the definition of the approach, the base functions used to process the strongly connected components is left open, allowing the definition of a wide range of functions to capture different semantics.

According to some analysis of basic properties which can be expected from any extension-based argumentation semantics [1], CF2 argumentation semantics is considered as the most accepted from the ones defined in terms of SCC-recursive semantics [2].

In this paper, following the recent results with respect to extension-based argumentation semantics in terms of logic programming semantics with negation as failure [5, 10, 15, 16, 18], we will show that not only extension-based argumentation semantics based on admissible sets can be characterised in terms of answer set models; but also SCC-recursive semantics such as the CF2 can be characterized in terms of answer set models. In particular, we will present a characterization of CF2 in terms of answer set models. This characterization suggests an approach for characterizing other extension-based argumentation semantics based on the SCC-recursive approach and an easy-to-use form for inferring the CF2 extensions of an argumentation framework.

The rest of the paper is structured as follows: In §2, a short presentation of basic concepts of answer set programming is presented. Also a short overview of extension-based argumentation semantics is presented. In §3, an approach for characterizing SCC-recursive semantics in terms of answer set models is presented. In particular, a characterization of CF2 is formalized. This section includes a subsection for illustrating the use of our results. In the last section, our conclusions are presented.

2. Background

In this section, we present a short presentation of basic concepts. Two main topics are covered: a) the syntax of logic programs, along with the definition of *Answer set semantics*; b) extension-based argumentation semantics, including an overview of Dung's and Baroni et al.'s approaches [2, 8].

2.1. Logic Programs

This subsection introduces the syntax of logic programs, along with a brief introduction to *Answer set semantics*.

2.1.1. Syntax

A signature $\mathcal{L}$ is a finite set of elements that we call atoms. A literal is an atom, a (positive literal), or the negation of an atom *not a* (negative literal). A disjunctive clause is a clause of the form: $a_1 \vee \ldots \vee a_m \leftarrow a_{m+1}, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$ where a_i is an atom, $1 \leq i \leq n$. When $n = m$ and $m > 0$, the disjunctive clause is an abbreviation of the fact $a_1 \vee \ldots \vee a_m \leftarrow \top$ where $\top$ is an atom that always evaluate to true. When $m = 0$ and $n > 0$ the clause is an abbreviation of $\bot \leftarrow a_1, \ldots, a_j, not\ a_{j+1}, \ldots, not\ a_n$ where $\bot$ is an atom that always evaluate to false. Clauses of this form are called constraints. A disjunctive logic program is a finite set of disjunctive clauses. A given set of disjunctive clauses $\{\gamma_1, \ldots, \gamma_n\}$ is also represented as $\{\gamma_1; \ldots; \gamma_n\}$ to avoid ambiguities with the use of comma in the body of the clauses.

We denote by $\mathcal{L}_P$ the signature of P, i.e., the set of atoms that occur in P. Given a signature $\mathcal{L}$, we write $Prog_{\mathcal{L}}$ to denote the set of all the programs defined over $\mathcal{L}$.

2.1.2. Answer Set Semantics

The following definition of an answer set for disjunctive logic programs was presented in [12]: Let P be any disjunctive logic program. For any set $S \subseteq \mathcal{L}_P$, let P^S be the logic program obtained from P by deleting: i) each clause that has a formula *not a* in its body with $a \in S$ ii) all formulæ of the form *not a* in the bodies of the remaining clauses.
Clearly P^S does not contain *not*, hence S is called an answer set of P if and only if S is a minimal model of P^S.

2.2. Extension-based argumentation semantics

This subsection introduces extension-based argumentation semantics starting from Dung's approach.

2.2.1. Dung's approach

In his work, Dung introduces the concept of argumentation framework which is defined as follows (All the definitions used in this subsection where taken from [8]):

Definition 1 *An argumentation framework is a pair* $AF = \langle AR, attacks \rangle$*, where* AR *is a set of arguments, and* attacks *is a binary relation on* AR*, i.e.* attacks $\subseteq AR \times AR$*. A attacks B (or B is attacked by A) if* $attacks(A, B)$ *holds.*

Following Dung's reading, A attacks B (or B is attacked by A) if $attacks(A, B)$ holds. A set S of arguments attacks B (or B is attacked by S) if B is attacked by an argument in S.

Definition 2

- *A set S of arguments is conflict-free if there are no arguments A, B in S such that A attacks B.*

- *An argument $A \in AR$ is acceptable with respect to a set S of arguments if and only if for each argument $B \in AR$: If B attacks A then B is attacked by S.*
- *A conflict-free set of arguments S is admissible if and only if each argument in S is acceptable w.r.t. S.*

From the extension-based argumentation semantics introduced in [8], we can identify two reasoning approaches: 1.- The extension-based argumentation semantics which follows a credulous reasoning approach; and 2.- The extension-based argumentation semantics which follows a sceptical reasoning approach. From the credulous semantics, one can mention, the stable, preferred and complete semantics and from the sceptical semantics, the grounded semantics. For presenting the results of this paper only the grounded extension is relevant. The grounded semantics is defined in terms of a *characteristic function*.

Definition 3 *Let the characteristic function, denoted by F_{AF}, of an argumentation framework $AF = \langle AR, attacks \rangle$ be defined as follows:*

$F_{AF} : 2^{AR} \rightarrow 2^{AR}$
$F_{AF}(S) = \{A |\ A \text{ is acceptable w.r.t. } S\ \}$

The grounded extension of an argumentation framework AF, denoted by GE_{AF}, is the least fixed point of F_{AF}

In his work [8], Dung suggests a general method for generating metainterpreters in terms of logic programming for argumentation systems. This approach is based in the following program:

Definition 4 *Given an argumentation framework $AF = \langle AR, attacks \rangle$, P_{AF} denotes the logic program defined by $P_{AF} = APU + AGU$ where*

$$APU = \{acc(x) \leftarrow not\ d(x) | x \in AR\} \cup \{d(x) \leftarrow attack(y,x), acc(y) | attack(y,x) \in attacks\}$$

and

$$AGU = \{attack(a,b) \leftarrow \top | (a,b) \in attacks\ , a \in AR, b \in AR\}$$

For each extension E of AF, $m(E)$ is defined as follows:

$$m(E) = AGU \cup \{acc(a) | a \in E\} \cup \{d(b) | b \text{ is attacked by some } a \in E\}$$

Based on P_{AF}, Dung was able to characterize the stable semantics and the grounded semantics in terms of logic programming semantics with negation as failure.

Theorem 1 *Let AF be an argumentation framework and E be an extension of AF. Then*

1. *E is a stable extension of AF if and only if $m(E)$ is an answer set of P_{AF}*

2. *E is a grounded extension of AF if and only if* $m(E) \cup \{not\ d(a) | a \in E\}$ *is the well-founded model [11] of* P_{AF}

This theorem is of relevance as it: a) defines a general method for generating meta-interpreters for argumentation systems, b) defines a general method for studying abstract argumentation semantics' properties in terms of logic programming semantics' properties.

2.2.2. SCC-recursiveness approach

The SCC-recursiveness approach is based on the notions of *path-equivalence* between nodes and *strongly connected components* (Please notice that, due to length restrictions, definitions in this subsection are presented informally. Formal definitions can be found in [2]).

Definition 5 *Given an argumentation framework* $AF = \langle AR, \text{attacks} \rangle$*, the binary relation of* path-equivalence *between nodes, denoted as* $PE_{AF} \subseteq (AR \times AR)$*, is defined as follows:*
— $\forall a \in AR$, $(a,a) \in PE_{AF}$,
— *given two distinct nodes a,* $b \in AR$, $(a,b) \in PE_{AF}$ *if and only if there is a path*[3] *from a to b and a path from b to a.*

Given an argumentation framework $AF = \langle AR, attacks \rangle$, the *strongly connected components* of AF are the equivalent classes of nodes which are defined according to the path-equivalence relation. The set of the strongly connected components of AF is denoted as $SCCS_{AF}$.

Definition 6 *Let* $AF = \langle AR, \text{attacks} \rangle$ *be an argumentation framework, and let* $S \subseteq AR$ *be a set of arguments. The* restriction *of AF to S is the argumentation framework* $AF \downarrow_S = \langle S, \text{attacks} \cap (S \times S) \rangle$.

Considering an argumentation framework, $AF = \langle AR, attacks \rangle$, a set $E \subseteq AR$ and a strongly connected component $S \in SCCS_{AF}$ the following sets can be defined: a) $D_{AF}(S,E)$ consists of the nodes of S attacked by E from outside S, b) $U_{AF}(S,E)$ consists of the nodes of S that are not attacked by E from outside S and are defended by E (i.e., their defeaters from outside S are all attacked by E), c) $P_{AF}(S,E)$ consists of the nodes of S that are not attacked by E from outside S and are not defended by E (i.e., at least one of their defeaters from outside S is not attacked by E), d) $UP_{AF}(S,E) = (S \setminus D_{AF}(S,E)) = (U_{AF}(S,E) \cup P_{AF}(S,E))$.

Here, we define $GF(AF,C)$ for an argumentation framework $AF = \langle AR, attacks \rangle$ and a set $C \subseteq AR$, representing the defended nodes of AF: two cases have to be considered in this respect.

If AF consists of exactly one strongly connected component, it does not admit a decomposition. On the other hand, if AF can be decomposed into several strongly connected components, then, $GF(AF,C)$ is obtained by applying recursively GF to each strongly connected component of AF, deprived of the nodes in $D_{AF}(S,E)$. Formally, this

[3] Given an argumentation framework $AF = \langle AR, attacks \rangle$ and $a, b \in AR$ there is a path between a and b if there is a sequence $\langle x_0, x_1, \ldots, x_n \rangle$ such that $\langle x_i, x_{i+1} \rangle \in attacks$ for $0 \leq i \leq n$ and $x_0 = a$ and $x_n = b$

means that for any $S \in SCCS_{AF}$, $(E \cap S) \in GF(AF \downarrow_{UP_{AF}(S,E)}, C')$, where C' represents the set of defended nodes of the restricted argumentation framework $AF \downarrow_{UP_{AF}(S,E)}$. The set C' can be determined by taking into account both the attacks coming from outside AF and those coming from other strongly connected components of AF.

Definition 7 *A given argumentation semantics S is* SCC-recursive *if and only if for any argumentation framework $AF = \langle AR, \text{attacks}\rangle$, $E_S(AF) = GF(AF, AR)$, where for any $AF = \langle AR, \text{attacks}\rangle$ and for any set $C \subseteq AR$, the function $GF(AF, C) \subseteq 2^{AR}$ is defined as follows: for any $E \subseteq AR$, $E \in GF(AF, C)$ if and only if*

- *in case $|SCCS_{AF}| = 1$, $E \in BF_S(AF, C)$,*
- *otherwise, $\forall S \in SCCS_{AF}(E \cap S) \in GF(AF \downarrow_{UP_{AF}(S,E)}, U_{AF}(S, E) \cap C)$.*

where $BF_S(AF, C)$ is a function, called base function, that, given an argumentation framework $AF = \langle AR, \text{attacks}\rangle$ such that $|SCCS_{AF}| = 1$ and a set $C \subseteq AR$, gives a subset of 2^{AR}.

Remark 1 *In the particular case where $BF_S(AF, C)$ is the function that returns the set of all maximal conflict-free sets of arguments $CF2$ is obtained.*

3. SCC-recursive semantics via Answer Sets

In this section, we are going to present our approach for characterizing SCC-recursive semantics in terms of answer set semantics. In particular, our approach is suitable for the class of SCC-recursive semantics defined by a conflict-free base function. This class of SCC-recursive semantics is of special interest since in [2] it was proved that any extension of these SCC-recursive semantics includes *the grounded extension*.

Like SCC-recursive semantics definition, our *declarative approach* for inferring SCC-recursive semantics is constructive and is based in three general steps: 1.-Inferring the grounded extension. 2.- Reducing the given argumentation framework by considering the inferred grounded extension. In this case, reduction is performed based on a labeling approach, labeling each argument as: *accepted*, *defeated* or *undefined*. 3.- If the reduced argumentation framework has undefined arguments, the base function is applied and the process follows in Step 1.

Observe that this approach infers the grounded semantics more than once. In order to manage the suggested approach, we define two general counters: one for inferring the grounded extension and another for controlling the general iterations suggested by Step 3.

We start by presenting a specification which characterizes the grounded semantics in terms of answer sets. In order to regard an argumentation framework as a logic program, we define the following programs: Given an argumentation framework $AF = \langle AR, attacks\rangle$,

$$\begin{aligned}
\Pi_{arg} = \quad & \{arg(a,0) \leftarrow \top | a \in AR\}.\\
\Pi_{at} = \quad & \{at(a,b) \leftarrow \top | (a,b) \in attacks\}.\\
\Pi_{int} = \quad & \{\, time(0) \leftarrow \top; \ldots; time(n+1) \leftarrow \top;\\
& \qquad int(0) \leftarrow \top; \ldots; int(n) \leftarrow \top; size(n) \leftarrow \top | n \text{ is the cardinality of } AR\}
\end{aligned}$$

Observe that essentially Π_{arg} and Π_{at} are mapping AF into predicates and Π_{int} is defining two counters and the number of arguments. The union of these three programs is denoted

by Π_{ini}. We want to clarify to the reader that in the following programs any instantiation of a variable N in *int(N)* will manage an iteration for inferring the grounded extension and any instantiation of a variable T in $time(T)$ will manage an iteration in the general process of inferring the SCC-recursive semantics.

Now let us introduce the following program:

$$
\begin{aligned}
\Pi_{GE} = \quad & a\_gr(X,0,T1) \leftarrow arg(X,T), T1 = T+1,\ not\ not\_a\_gr(X,T)\cdot \\
& not\_a\_gr(X,T) \leftarrow arg(X,T), at(Y,X),\ not\ d(Y,T), time(T)\cdot \\
& a\_gr(X,N,T) \leftarrow int(N), N > 0, arg(X,T),\ not\ not\_a\_gr\_d(X,N,T), time(T)\cdot \\
& not\_a\_gr\_d(X,N,T) \leftarrow at(Y,X), N = M+1,\ not\ attacked(Y,M,T), arg(X,T), \\
& \qquad arg(Y,T), time(T)\cdot \\
& attacked(Y,M,T) \leftarrow arg(Y,T), arg(Z,T), at(Z,Y), a\_gr(Z,M,T), time(T)\cdot
\end{aligned}
$$

This program is defining a characterization of the grounded semantics in terms of answer set models, as formalized in the following proposition:

Proposition 1 *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework and $E \subseteq AR$. E is the grounded extension of AF if and only if M is an answer set of $\Pi_{ini} \cup \Pi_{GE}$ such that $E = \{x | a\_gr(x,N,T) \in M\}$.*

Proof: **(sketch)** Let M be an answer set of $\Pi_{ini} \cup \Pi_{GE}$, $n = |AR|$, $S_0 = \{x | a\_gr(x,0,\_) \in M\}$ and $S_i = \{x | a\_gr(x,i,\_) \in M, 1 \leq i \leq n\}$. Hence, the proof follows by induction and the following observations:

1. $S_{i-1} \subseteq S_i (1 \leq i \leq n)$ and S_i is an admissible set.
2. $S_i (1 \leq i \leq n)$ characterizes the characteristic function F_{AF}.
3. Since $S_i (1 \leq i \leq n)$ is monotonic, it reaches a fix-point.

■

Given a grounded extension, one can define different states of an argument: 1.- an argument which belongs to the grounded extension can be considered as accepted (ac_gr), 2.- an argument which is attacked by an accepted argument can be considered as defeated (d_gr), and 3.- an argument which is neither accepted nor defeated can be considered undefined (i_gr). These states of the argument are captured by the following program:

$$
\begin{aligned}
\Pi_{gr\_states} = \quad & ac\_gr(X,T) \leftarrow a\_gr(X,fixp,T), time(T), size(fixp)\cdot \\
& d\_gr(X,T) \leftarrow at(Y,X), ac\_gr(Y,T), arg(Y,T), arg(X,T), time(T)\cdot \\
& i\_gr(X,T) \leftarrow arg(X,T),\ not\ ac\_gr(X,T),\ not\ d\_gr(X,T), time(T)\cdot \\
& at\_d(X,Y,T) \leftarrow arg(X,T), arg(Y,T), at(X,Y), \\
& \qquad not\ not\_at\_d(X,Y,T), time(T)\cdot \\
& not\_at\_d(X,Y,T) \leftarrow arg(X,T), arg(Y,T), at(X,Y), d\_gr(X,T), time(T)\cdot
\end{aligned}
$$

Observe that the predicate $at\_d(X,Y,T)$ captures the arguments which are attacked by defeated arguments. Also observe that the predicate $ac\_gr(X,T)$ defines a subset of an extension of SCC-recursive semantics.

Proposition 2 *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework and S be a SCC-recursive semantics such that each extension in S contains the grounded extension. If $E \in S(AF)$, then an answer set M of $\Pi_{ini} \cup \Pi_{GE} \cup \Pi_{gr\_states}$ exists, such that $\{x | ac\_gr(x,T) \in M\} \subseteq E$ and $\{x | d\_gr(x,T) \in M\} \nsubseteq E$*

Proof: **(sketch)** Let M be an answer set of $\Pi_{ini} \cup \Pi_{GE} \cup \Pi_{gr\_states}$, $S_{ac\_gr}(M) = \{x | ac\_gr(x,\_) \in M\}$ and $S_{d\_gr}(M) = \{x | d\_gr(x,\_) \in M\}$. Let us denote by $ASP(P)$, the answer set models of a given logic program P.

Observations:

1. By Proposition 1, if $M \in ASP(\Pi_{ini} \cup \Pi_{GE} \cup \Pi_{gr\_states})$, then $S_{ac\_gr}(M)$ is the grounded extension of AF.
2. $S_{ac\_gr}(M) \cap S_{d\_gr}(M) = \emptyset$.

If S is a SCC-recursive semantics such that each extension in S contains the grounded extension, then $GE_{AF} \in \bigcap_{E \in S(AF)} E$. By Observation 1, $GE_{AF} = S_{ac\_gr}(M)$. Then $\forall E \in S(AF), S_{ac\_gr}(M) \in E$. Therefore, by Observation 2, $\forall E \in S(AF), S_{d\_gr}(M) \notin E$.

■

This proposition suggests that in the construction of an extension of a SCC-recursive semantics one can consider an argumentation framework restricted to undefined arguments, *i.e.*, $\{a | i\_gr(a,T) \in M\}$. This idea of recursion follows Definition 7 where the recursive step is restricted to $AF \downarrow_{UP_{AF(S,E)}}$. Observe that in order to define this restricted (*w.r.t.* undefined arguments) argumentation framework, one has to identify *the strongly connected components* that exist in the argumentation framework and follow the *directionality principle*[4].

$$
\begin{aligned}
\Pi_{AF\downarrow_{i\_gr}} = \quad & base(X,T) \vee other\_base(X,T) \leftarrow i\_gr(X,T), time(T)\cdot \\
& \leftarrow base(X,T), base(Y,T), X! = Y, not\ cycle(X,Y,T), time(T)\cdot \\
& \leftarrow base(X,T), ar(Y), X! = Y, other\_base(Y,T), cycle(X,Y,T)\cdot \\
& \leftarrow not\_base(T), time(T)\cdot \\
& \leftarrow incomplete(T), time(T)\cdot \\
& incomplete(T) \leftarrow i\_gr(X,T), not\ not\_empty(T), time(T)\cdot \\
& not\_empty(T) \leftarrow base(X,T), time(T)\cdot \\
& not\_base(T) \leftarrow base(X,T), arg(Y,T), other\_base(Y,T), at(Y,X), time(T)\cdot \\
& cycle(X,Y,T) \leftarrow path(X,Y,T), path(Y,X,T), time(T)\cdot \\
& path(X,Y,T) \leftarrow at(X,Y), arg(X,T), arg(Y,T), time(T)\cdot \\
& path(X,Y,T) \leftarrow arg(X,T), arg(Y,T), arg(Z,T), at(X,Z), \\
& \qquad path(Z,Y,T), time(T)\cdot
\end{aligned}
$$

Observe that in this program we are using predefined predicates such as $!=$ which are common in answer set solvers such as the DLV solver [7]. Once we have identified our restricted argumentation framework, the status of the arguments (identified as accepted or defeated) have to be preserved.

$$
\begin{aligned}
\Pi_{inertial} = \quad & ac(X,T1) \leftarrow T1 = T+1, ac(X,T), time(T1)\cdot \\
& d(X,T1) \leftarrow T1 = T+1, d(X,T), time(T1)\cdot \\
& ac(X,T) \leftarrow ac\_gr(X,T), time(T)\cdot \\
& d(X,T) \leftarrow d\_gr(X,T), time(T)\cdot \\
& arg(X,T1) \leftarrow arg(X), T1 = T+1, not\ ac(X,T), notd(X,T), time(T), time(T1)\cdot
\end{aligned}
$$

Observe that the last clause of this program is defining the set of arguments that have to be considered in the next iteration of the process.

In order to characterize the CF2 argumentation semantics which is a SCC-recursive semantics, we define a base function which infers maximal conflict free sets of arguments by taking into account the directionality principle.

[4] *Directionality principle*: Nodes defeated by an extension E play no role in the selection of nodes to be includes in E.

$$
\begin{aligned}
\Pi_{base\_function} = \quad & d(X,T) \leftarrow ac(Y,T), at(Y,X), time(T)\cdot \\
& ac(X,T) \leftarrow base(X,T), not\ d(X,T), time(T)\cdot \\
& d(X,T) \leftarrow at(Y,X), base(X,T), base(Y,T), ac(Y,T), time(T)\cdot \\
& d(X,T) \vee d(Y,T) \leftarrow at(Y,X), path\_d(X,Y,T), base(X,T), \\
& \qquad base(Y,T), time(T)\cdot \\
& path\_d(X,Y,T) \leftarrow at(X,Y), base(X,T), base(Y,T), time(T)\cdot \\
& path\_d(X,Y,T) \leftarrow at(X,Z), base(X,T), base(Y,T), base(Z,T), \\
& \qquad path\_d(Z,Y,T), time(T)\cdot \\
& accepted(X) \leftarrow ac(X,fixp), size(fixp)\cdot
\end{aligned}
$$

For connecting Π_{GE} with $\Pi_{base\_function}$, we define the program $\Pi_{GE'}$ as the program Π_{GE} plus the following rule:

$$a\_gr(X,0,T1) \leftarrow T1 = T+1, ac(X,T)$$

Consider the program P_{CF2} which is defined as follows: Given an argumentation framework AF

$$P_{CF2}(AF) = \Pi_{ini} \cup \Pi_{GE'} \cup \Pi_{gr\_states} \cup \Pi_{AF\downarrow_{i\_gr}} \cup \Pi_{inertial} \cup \Pi_{base\_function}$$

The following theorem shows a characterization of CF2 argumentation semantics in terms of answer set models of the P_{CF2} program.

Theorem 2 *Let $AF = \langle AR, attacks\rangle$ be an argumentation framework. $E \in CF2(AF)$ if and only if there exists an answer set M of $P_{CF2}(AF)$ such that $E = \{a | accepted(a) \in M\}$.*

Proof: **(sketch)** The proof follows by Propositions 1, 2 and the following observations:

1. The programs Π_{ini} and $\Pi_{base\_function}$ characterize the maximal conflict-free sets of a given argumentation framework.
2. The program $\Pi_{inertial}$ characterizes the strongly connected components of a given argumentation framework.
3. The programs $\Pi_{inertial}$ and $\Pi_{AF\downarrow_{i\_gr}}$ induce the directionality principle.

■

We want to point out at least two implications of this characterization of CF2:

1. By changing the behavior of $\Pi_{base\_function}$ one can characterize different SCC-recursive semantics.
2. This characterization of CF2 suggests an easy-to-use form for inferring extensions of SCC-recursive semantics such as CF2 extensions.

3.1. Applications of Theorem 2

We can find different strategies for computing extension-based argumentation semantics [3, 6, 9, 14]; however, a common point among these approaches is to address some of the following questions[5] *w.r.t.* a given extension-based argumentation semantics S, an argumentation framework $AF = \langle AR, attacks\rangle$ and $A \in AR$:

[5]Notice, questions 2,3 and 6 answer 'credulous acceptance problem *w.r.t. S*', 'skeptical acceptance problem *w.r.t. S*' and 'skeptical refusal problem *w.r.t. S*' respectively

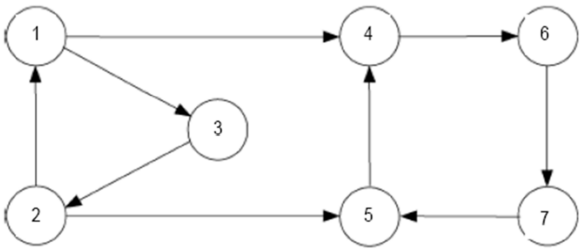

Figure 1. Graph representation of the argumentation framework $AF = \langle\{1,2,3,4,5,6,7\},\{(1,4),(1,3),(2,1),(2,5),(3,2),(4,6),(6,7),(7,5),(5,4)\}\rangle$

1. Which are the extensions in $S(AF)$?
2. Is A contained in an extension of $S(AF)$?
3. Is A contained in all the extension of $S(AF)$?
4. Which are all the extensions containing A?
5. Which are the extensions that attacks A?
6. Is A attacked by all the extensions of $S(AF)$?

One of the main applications of Theorem 2 is that one can take advantage of efficient answer set solvers, *e.g.*, [7], for answering the given questions *w.r.t.* SCC-recursive semantics such as CF2. For instance, the DLV system allows the use of different front-ends for performing different kinds of queries *w.r.t.* the answer set models of a given logic programs; this means that by using these front-ends one can answer the given questions *w.r.t.* S, AF and A. In order to illustrate these applications of Theorem 2, let AF be the argumentation framework of Figure 1 and `pcf2.dlv` be the program which contains $P_{CF2}(AF)$. Now we are going to answer each of the given questions.

Which are the extensions in $CF2(AF)$? For answering this question, let us call DLV with the filter *accepted*:

```
$ dlv pcf2.dlv -filter=accepted
{accepted(2), accepted(4), accepted(7)}{accepted(1), accepted(5),
accepted(6)}{accepted(3), accepted(4), accepted(7)}{accepted(3),
accepted(5), accepted(6)}
```

This means that the provided set of sets of arguments are all the possible CF2 extensions of the framework.

Is 5 contained in an extension of $CF2(AF)$? For answering this question, let `query1` be the file: `accepted(5)?` Now let us call DLV with the filter *accepted*, the *brave/credulous reasoning* front-end and `query1`:

```
$ dlv pcf2.dlv -brave -filter=accepted query1
accepted(5) is bravely true, evidenced by {accepted(3), accepted(5),
accepted(6)}
```

This means that it is true that the argument 5 belongs to a CF2 extension and even more we have a CF2 extension which contains the argument 5.

Is 5 contained in all the extension of $CF2(AF)$? For answering this question, let `query2` be the file: `accepted(5)?` Now let us call DLV with the filter *accepted*, the

cautious/skeptical reasoning front-end and `query2`:

```
$ dlv pcf2.dlv -cautious -filter=accepted query2
accepted(5) is cautiously false, evidenced by {accepted(2), accepted(4),
accepted(7)}
```

This means that it is false that the argument 5 belongs to all CF2 extensions and even more, DLV provides a CF2 extension that serves as counterexample

Which are all the extensions containing 3? For answering this question, let `query3` be the file: `accepted(3).` ? Now let us call DLV with the filter *accepted* and `query3`:

```
$ dlv pcf2.dlv -filter=accepted query3s
{accepted(3), accepted(5), accepted(6)}{accepted(3), accepted(4),
accepted(7)}
```

This means that the provided set of sets of arguments are all the possible CF2 extensions on the framework containing 3.

In order to manage the questions *w.r.t.* attacks, let `pcf2-at.dlv` be the program which contains $P_{CF2}(AF) \cup \{attacked(X) \leftarrow at(Y,X), accepted(Y)\}$.

Which are the extensions that attack 3**?** For answering this question, let `query4` be the file: `attacked(3).` ? Now let us call DLV with the filter *accepted* and `query4`:

```
$ dlv pcf2-at.dlv -filter=accepted query4
{accepted(1), accepted(5), accepted(6)}
```

This means that the provided set of arguments is all the possible CF2 extensions on the framework containing arguments attacking 3.

Is 5 **attacked by all the extensions of** $CF2(AF)$**?** For answering this question, let `query5` be the file: `attacked(5).` ? Now let us call DLV with the filter *accepted*,the *cautious/skeptical reasoning* front-end and `query5`:

```
$ dlv pcf2-at.dlv -cautious -filter=accepted query5
attacked(3) is cautiously false, evidenced by {accepted(3), accepted(5),
accepted(6)}
```

This means that it is false that the argument 5 is attacked by all CF2 extensions and even more, DLV provides a CF2 extension that serves as counterexample

4. Conclusions

The study and understanding of extension-based argumentation semantics have shown to be a crucial point in argumentation theory [1, 2, 8, 17]. This is mainly because they capture several approaches for performing argumentation reasoning. To find relationships between them and well-acceptable approaches such as answer set semantics helps in the exploration and implementation of prominent non-monotonic reasoning approaches.

So far we have already shown that extension-based argumentation semantics based on admissible sets are able to be characterized in terms of answer set models [5, 8, 10, 15, 16, 18]; however, it is well-known that they have unexpected behaviors [2, 17]. Hence, the consideration of emerging approaches seems to be crucial. The emerging approaches for defining new extension-based argumentation semantics is really diverse; however, ap-

proaches such as the ones based on SCC-recursive semantics looks sound. One can find SCC-recursive semantics such as CF2 which converges with extension-based argumentation semantics based on logic programming semantics with negation as failure [16].

Now in this paper, we have shown that not only extension-based argumentation semantics based on admissible sets can be characterized in terms of answer set models; but also extension-based argumentations semantics based on strongly connected components can be characterized in terms of answer set models. This result opens the possibility of exploring new SCC-recursive semantics which can be interpreted in a straightforward form in terms of answer set models. And of course, the possibility of implementing fast prototypes of them using answer set programming platforms [13].

References

[1] P. Baroni and M. Giacomin. On principle-based evaluation of extension-based argumentation semantics. *Artificial Intelligence.*, 171(10-15):675–700, 2007.

[2] P. Baroni, M. Giacomin, and G. Guida. SCC-recursiveness: a general schema for argumentation semantics. *Artificial Intelligence*, 168:162–210, October 2005.

[3] P. Besnard and S. Doutre. Checking the acceptability of a set of arguments. In *Tenth International Workshop on Non-Monotonic Reasoning (NMR 2004),*, pages 59–64, June 2004.

[4] M. Caminada. Semi-Stable semantics. In P. E. Dunne and T. J. Bench-Capon, editors, *Proceedings of COMMA*, volume 144, pages 121–130. IOS Press, 2006.

[5] J. L. Carballido, J. C. Nieves, and M. Osorio. Inferring Preferred Extensions by Pstable Semantics. *Iberoamerican Journal of Artificial Intelligence (Inteligencia Artificial) ISSN: 1137-3601*, 13(41):38–53, (doi: 10.4114/ia.v13i41.1029), 2009.

[6] C. Cayrol, S. Doutre, and J. Mengin. On Decision Problems related to the preferred semantics for argumentation frameworks. *Journal of Logic and Computation*, 13(3):377–403, 2003.

[7] S. DLV. Vienna University of Technology. http://www.dbai.tuwien.ac.at/proj/dlv/, 1996.

[8] P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2):321–358, 1995.

[9] P. M. Dung, P. Mancarella, and F. Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171(issues 10-15):642–674, 2007.

[10] U. Egly, S. A. Gaggl, and S. Woltran. Aspartix: Implementing argumentation frameworks using answer-set programmin. In M. G. de la Banda and E. Pontelli, editors, *International Conference of Logic Programming (ICLP)*, volume 5366 of *Lecture Notes of Computer Science*, pages 734–738. Springer, 2008.

[11] A. V. Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[12] M. Gelfond and V. Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[13] I. Gómez-Sebastià and J. C. Nieves. WizArg: Visual Argumentation Framework Solving Wizard. In *In Proceedings of CCIA'2010*. Frontiers in Artificial Intelligence and Applications, IOS Press, accepted.

[14] S. Modgil and M. Caminada. *Argumentation in Artificial Intelligence*, chapter Proof Theories and Algorithms for Abstract Argumentation Frameworks, pages 105–129. Springer, 2009.

[15] J. C. Nieves, M. Osorio, and U. Cortés. Preferred Extensions as Stable Models. *Theory and Practice of Logic Programming*, 8(4):527–543, July 2008.

[16] J. C. Nieves, M. Osorio, and C. Zepeda. A Schema for Generating Relevant Logic Programming Semantics and its Applications in Argumentation Theory. *Fundamenta Informaticae*, accepted.

[17] H. Prakken and G. A. W. Vreeswijk. Logics for defeasible argumentation. In D. Gabbay and F. Günthner, editors, *Handbook of Philosophical Logic*, volume 4, pages 219–318. Kluwer Academic Publishers, Dordrecht/Boston/London, second edition, 2002.

[18] K. N. Toshiko Wakaki. *Computing Argumentation Semantics in Answer Set Programming*, volume 5447/2009 of *Lecture Notes in Computer Science*, pages 254–269. 2009.